An ad-hoc and monolithic method for ensuring that corecursive definitions are productive

Nils Anders Danielsson

Nottingham

AIM 9, 2008-11-27

Hamming numbers (almost)

An ordered stream of all products of 2 and 3:

$$\mathit{hamming} \;=\; 1 : \mathit{merge}\ (\mathit{map}\ (2\ *)\ \mathit{hamming})\ (\mathit{map}\ (3\ *)\ \mathit{hamming})$$

- Productive?
- How can we get Agda to believe that it is?

One method

1. Define problem-specific language.
2. Implement provably productive interpreter.

The implementation can take advantage of the host language's productivity checker.

Disclaimer:
Hopefully this method will soon become obsolete.

…because it is awkward to use in practice.

However:

- Interesting to see what can be done without adding new features.
- Flexible.

How does it work?

Back to the example

```
codata Stream (A : Set) : Set where
  _≺_ : A → Stream A → Stream A
```

Hamming numbers again

```
hamming : Stream ℕ
hamming ∼ 1 ≺ merge (map (_*_ 2) hamming)
                    (map (_*_ 3) hamming)
```

- Not guarded by constructors.
- But what if *merge* and *map* were constructors?

Ad-hoc programming language

```
mutual
  codata WHNF : Set → Set1 where
    _≺_ : forall {A} → A → Prog (Stream A) →
          WHNF (Stream A)
  data Prog : Set → Set1 where
    ↓_    : forall {A} → WHNF A → Prog A
    map   : forall {A B} → (A → B) →
            Prog (Stream A) → Prog (Stream B)
    merge : Prog (Stream ℕ) →
            Prog (Stream ℕ) →
            Prog (Stream ℕ)
```

Guarded definition

```
hamming : Prog (Stream ℕ)
hamming ∼ ↓ 1 ≺ merge (map (_*_ 2) hamming)
                      (map (_*_ 3) hamming)
```

- Guarded by constructors.
- _≺_ is a coconstructor.
- Note: Corecursive definition of inductive value.

Interpreter

1. One-step evaluator:

   ```
   whnf : forall {A} → Prog A → WHNF A
   ```

   Recursive: WHNF always reached in finite time.
2. Full evaluation:

   ```
   value : forall {A} → WHNF A → A
   value (x ≺ prog) ∼ x ≺ value (whnf prog)
   ⟦_⟧ : forall {A} → Prog A → A
   ⟦ prog ⟧ = value (whnf prog)
   ```

   Uses guarded corecursion.

One-step evaluator

Structurally recursive:

```
whnf : forall {A} → Prog A → WHNF A
whnf (↓ w) = w
whnf (map f xs) with whnf xs
... | x ≺ xs′ = f x ≺ map f xs′
whnf (merge xs ys) with whnf xs | whnf ys
... | x ≺ xs′ | y ≺ ys′ with cmp x y
... | lt = x ≺ merge xs′ ys
... | eq = x ≺ merge xs′ ys′
... | gt = y ≺ merge xs ys′
```

Wrapping up

$$
\begin{aligned}
&\mathit{ham} \;:\; \mathit{Stream}\ \mathbb{N} \\
&\mathit{ham} \;=\; [\![\ \mathit{hamming}\ ]\!]
\end{aligned}
$$

Perhaps one should also prove that *ham* satisfies its intended defining equation.

What happens with unproductive code?

Productivity ⇒ termination

Productivity problems are sometimes turned into termination problems:

$$
\begin{aligned}
&\mathit{map}_2 \;:\; \textbf{forall}\ \{A\ B\} \to (A \to B) \to \\
&\qquad\quad \mathit{Prog}\ (\mathit{Stream}\ A) \to \mathit{Prog}\ (\mathit{Stream}\ B) \\
&\mathit{map}_2\ f\ (x \prec x' \prec \mathit{xs}'') \sim f\ x \prec f\ x' \prec \mathit{map}_2\ f\ \mathit{xs}''
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{hamming} \;:\; \mathit{Stream}\ \mathbb{N} \\
&\mathit{hamming} \sim 1 \prec \mathit{merge}\ (\mathit{map}_2\ (\_*\_\ 2)\ \mathit{hamming}) \\
&\qquad\qquad\qquad\qquad\qquad\quad (\mathit{map}_2\ (\_*\_\ 3)\ \mathit{hamming})
\end{aligned}
$$

Productivity ⇒ termination

Productivity problems are sometimes turned into termination problems:

$$
\begin{aligned}
&\textbf{data}\ \mathit{Prog} \;:\; \mathit{Set} \to \mathit{Set1}\ \textbf{where} \\
&\quad \mathrm{map}_2 \;:\; \textbf{forall}\ \{A\ B\} \to (A \to B) \to \\
&\qquad\qquad \mathit{Prog}\ (\mathit{Stream}\ A) \to \mathit{Prog}\ (\mathit{Stream}\ B)
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{whnf}\ (\mathrm{map}_2\ f\ \mathit{xs})\ \textbf{with}\ \mathit{whnf}\ \mathit{xs} \\
&\ldots \mid x \prec \mathit{xs}'\ \textbf{with}\ \mathit{whnf}\ \mathit{xs}' \\
&\ldots \mid x' \prec \mathit{xs}'' \;=\; f\ x \prec (\downarrow f\ x' \prec \mathrm{map}_2\ f\ \mathit{xs}'')
\end{aligned}
$$

How far can this be taken?

Flexibility

It is possible to handle map_2:

$$
\begin{aligned}
&\textbf{mutual} \\
&\quad \textbf{data}\ \mathit{WHNF}_2 \;:\; \mathit{Set} \to \mathit{Set1}\ \textbf{where} \\
&\qquad \langle\_,\_\rangle\!\prec\_ \;:\; \textbf{forall}\ \{A\} \to \\
&\qquad\qquad A \to A \to \mathit{Prog}_2\ (\mathit{Stream}\ A) \to \\
&\qquad\qquad \mathit{WHNF}_2\ (\mathit{Stream}\ A)
\end{aligned}
$$

Flexibility

It is possible to handle map_2:

```
data Prog2 : Set → Set1 where
  ↓_   : forall {A} →
         WHNF2 A → Prog2 A
  map2 : forall {A B} →
         (A → B) →
         Prog2 (Stream A) → Prog2 (Stream B)
```

Flexibility

It is possible to handle map_2:

```
whnf2 : forall {A} → Prog2 A → WHNF2 A
whnf2 (↓ w) = w
whnf2 (map2 f xs) with whnf2 xs
... | ⟨ x , x′ ⟩≺ xs″ = ⟨ f x , f x′ ⟩≺ map2 f xs″
```

Flexibility

- Can be generalised from 2 to larger depths.
- Functions like *tail* can be handled.
  (But a coercion constructor may be necessary.)
- Can handle other types as well.
  - Breadth-first labelling of potentially infinite trees.

Equality proofs also possible

Unique fixed-points ⇒ guarded coinduction:

```
iterate-fusion h f1 f2 hyp x ∼
  map h (iterate f1 x)
    ≡⟨ ≡-refl ⟩
  ↓ h x ≺ map h (iterate f1 (f1 x))
    ≈⟨ ↓ ≡-refl ≺ iterate-fusion h f1 f2 hyp (f1 x) ⟩
  ↓ h x ≺ iterate f2 (h (f1 x))
    ≡⟨ ≡-cong (\y → ⟦ ↓ h x ≺ iterate f2 y ⟧)
              (hyp x) ⟩
  ↓ h x ≺ iterate f2 (f2 (h x))
    ≡⟨ ≡-refl ⟩
  iterate f2 (h x)
    □
```

What about the drawbacks?

Drawbacks

- Ad-hoc.
- Monolithic.
- Awkward.
- Limited support for higher-order functions:
  $(Prog\ A \rightarrow Prog\ B) \rightarrow \ldots$ is negative.
- Inefficient: sharing lost.

Sharing lost

```
_,_ : forall {A B} →
      WHNF A → WHNF B → WHNF (A × B)

fst : forall {A B} → Prog (A × B) → Prog A

whnf (fst prog) with whnf prog
... | (x,y) = x
```

- Can perhaps be worked around by implementing a call-by-need interpreter…

Conclusion

- Fun to play around with…
- …but for real work we need something more convenient.
- What? (Andreas Abel might add to the discussion tomorrow.)